Summary:

CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo; Yale University

Keywords :

- User concurrency, I/O concurrency and Multicore concurrency
- MCS locks, Multicore Shared Memory Systems
- Starvation-freedom, Liveness
- Formal verification, Certified Concurrent Layers, Coq

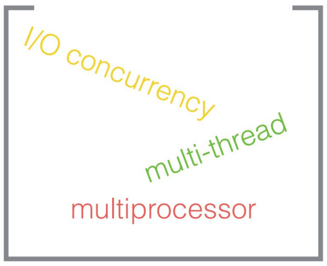
Motivation for Verified Concurrent OS Kernel :

- Concurrent Operating Systems are backbone of all systems including safety-critical software systems in the world.
- Any error inside the kernel can lead to bigger disaster.
- Complete formal verification is the only known way to guarantee that a system is free of programming errors.[seL4 -- SOSP'09]

Limitations of Prior work :

- Prior work has formally verified the functional correctness of sequential kernels, file systems, and device drivers, but none of these systems have addressed the important issues of concurrency.
- The seL4 team was the first to verify the functional correctness and security properties of a high- performance L4-family microkernel, however seL4 does not support Multicore shared memory concurrency with fine-grained locking.
- Xu et al. have successfully verified key modules in the µC/OS-II kernel which supports preemption but only on a single-core machine. They have not verified any assembly code nor connected their verified C- like source programs to any certified compiler so there is no end-to-end theorem about the entire kernel.

Why formal verification of Concurrent OS is hard :



- Concurrent kernels allow interleaved execution of kernel/user modules across different abstraction layers;

they contain many interdependent components that are difficult to untangle.

- Checking functional correctness of thread yield/sleep/wakeup primitives or interrupts to switch control and support synchronization or Multicore concurrency with fine-grained locking is intractable, and even if it is possible, its cost would far exceed that of verifying a single-core sequential kernel.
- Proving liveness i.e. system calls eventually return is very hard as this depends on the progress of the concurrent primitives.
- Providing extensibility (kernel plug-in) support requires to encapsulate interference, otherwise even a small edit could incur huge verification overhead.

Major Goals:

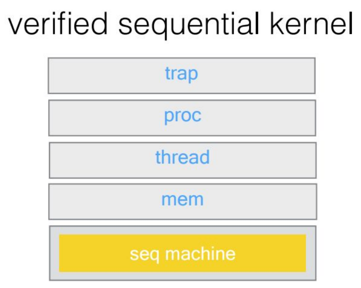
- Verification should not impose significant overhead on kernel performance.
- Should be able to prove global properties of user-level processes and virtual machines built on top of certified kernel.
- Extensibility support for new kernel Abstractions and Processes, i.e. it must support transfer of global properties proved at a high abstraction level down to any lower abstraction level which will also minimize the cost of development and maintenance.
- Certified kernel rather than verified kernel(machine-checkable proof).

How CertiKOS Architecture solves the verification problem:

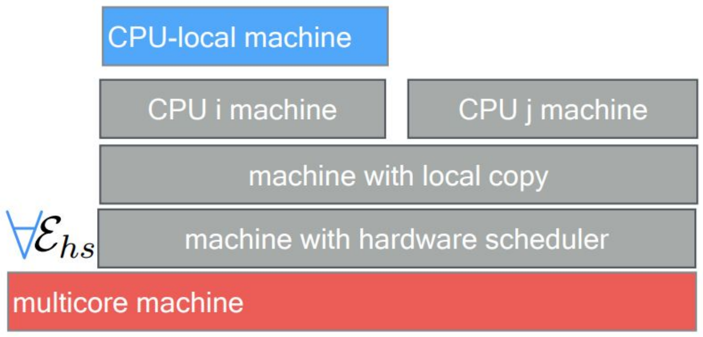
- kernel as composition of various certified concurrent abstraction layers.
- The environment context of a kernel K could be other kernel threads on same CPU or a copy of K running on another CPU due to shared-memory concurrency.
- Use of environment context at each layer and applying techniques for verifying sequential programs to verify functional correctness of concurrent programs also.
- Temporal invariants(e.g. Fair OS Scheduler) over these environment contexts to prove liveness.

Design Contributions:

- **certified concurrent sequential layers** (L1,M,L2) and a mechanized proof object showing that the layer implementation M, built on top of the interface L1 (the underlay), is a contextual refinement of the desirable interface L2 (the overlay).



- **Multicore hardware model**
  - Arbitrary interleavings at the level of assembly instructions.
  - Memory accesses are divided into Private(**local objects), Synchronized shared memory accesses (Atomic Objects)** with a **logical log** to maintaining the entire history of the operations that were performed on the object during an execution.
- **Hardware scheduler (εhs)**
  - It specifies a particular interleaving for an execution resulting in a deterministic machine model



- **Push/Pull Model**
  - Each CPU maintains local copy of shared memory blocks.
  - The pull operation over a particular memory block updates a CPU's local copy of that block to be equal to the one in the shared memory, marking the local block as valid and the shared version as invalid.
  - The push operation updates shared version to be equal to the local block, marking the shared version as valid and the local block as invalid.
  - Among each shared memory block and all of its local copies, only one can be valid at any single moment of machine execution.

- **Partial machine with environment context**
  - The partial machine model is configured with an active CPU set and it queries the environment context whenever it reaches a switch point that attempts to switch to a CPU outside the active set.
  - Each environment context takes the current log and returns a list of events from the context programs (i.e., those outside of Active CPU set).
  - The response function simulates the observable behavior of the context CPUs and imposes some invariants over the context.
  - The hardware scheduler is also a part of the environment context.

- **CPU-local machine model**
    - There is a switch point before each instruction, leading to unnecessary interleavings (e.g., those between private operations).
    - In CPU-local machine model for a CPU i, switch points only appear before atomic or push/pull operations.
    - The switch points before shared or private operations are removed via : shuffling/delaying and merging.

Limitations :

- The Certified mC2 kernel is not as comprehensive as real-world kernels eg Linux.
- The underlying assembly machine assumes strong sequential consistency for all atomic instructions, not the x86 TSO consistency and only covers a small part of the full x86 instruction set.
- Any code for TLB shootdown is not modeled and hence cannot be verified.
- Lacks a certified storage system.
- The CompCertX assembler, Bootloader, PreInit module (which initializes the CPUs and devices), and ELF loader are unverified and assumed to be correct.

Check your understanding :

- Standard Mesa-style condition variables do not guarantee starvation-freedom. How can this be fixed by using a FIFO queue of condition variables.
- For each environmental context, if A is singleton, the thread modular Machine behave like sequential machine.
- How can we guarantee that P is data-race free, by showing that a program P is safe (never goes wrong) on Machine with local copy of shared memory for all possible hardware schedulers.
- Let C be the entire CPU set, show that EC(partial machine,C) = EC(hs).
- Why all the query results from the environment context before shared and private operations can be shifted just before the next atomic or push/pull operation.